ДМ. Теормин №2

arslee.me

Boolean Algebra

Boolean function and Boolean formula

Булева функция: $f : \{0,1\}^n \to \{0,1\}$

Булева формула описывает булеву функцию. Состоит из переменных, констант 0/1 и операторов.

Number of n-ary Boolean functions

Существует 2^{2^n} функций от n переменных.

Minterm and maxterm

- Минтерм - куб, содержащий все n переменных.
- Макстерм - клоз, содержащий все n переменных.

DNF and CNF

- ДНФ: дизъюнкция кубов: $ab + \overline{c}d$
- КНФ: конъюнкция клозов: $(a + b)(\overline{c} + d)$

Negation normal form (NNF)

Формула находится в ННФ, если в ней используются только операции $\neg$, $\vee$, $\wedge$, при этом инверсия находится только над литералами.

Пример:
- $\overline{a \wedge b}$ - не ННФ
- $a \wedge \left(\overline{b} \oplus c\right)$ - не ННФ
- $\overline{a} \wedge (b \vee \overline{c})$ - ННФ

Shannon expansion and cofactors

Пусть дано $f(x, y_1, ..., y_n)$.

Кофакторы f по x - функции вида:
- $f_0(y_1, ..., y_n) = f(0, y_1, ...y_n)$
- $f_1(y_1, ..., y_n) = f(1, y_1, ...y_n)$

Разложение Шеннона:

$$f(x, y_1, ...y_n) = \overline{x} \cdot f_0(y_1, ..., y_n) + x \cdot f_1(y_1, ..., y_n)$$

Algebraic normal form (ANF) / Zhegalkin polynomial

АНФ (или полином Жегалкина) - представление функции в виде XOR'а конъюнкций переменных и константы 0/1.

Пример:

$$f(x, y, z) = xy \oplus xz \oplus y \oplus 1$$

По сути это многочлен над конечным полем $\mathbb{F}_2$:
- $\oplus$ - сумма
- $\wedge$ - умножение
- $0, 1$ - коэффициенты

Главная особенность: каждая Булева функция имеет *единственное* представление в виде АНФ.

Methods for ANF construction

1. Метод неопределенных коэффициентов.

   Для каждой строки таблицы истинности решаем линейные уравнения по коэффициентам.

2. Метод треугольника.

   Первый столбец - значения из таблицы истинности, следующие столбцы - XOR левого и левого нижнего ячеек. Верхняя строка треугольника будет соответствовать включению члена в полином:

A	B	C	P
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

000	001	010	011	100	101	110	111
1	C	B	BC	A	AC	AB	ABC
①	①	0	0	0	0	①	0
0	1	0	0	0	1	1	
1	1	0	0	1	0		
0	1	0	1	1			
1	1	1	0				
0	0	1					
0	1						
1							

$P = 1 \oplus C \oplus AB$

3. Метод карт Карно.

   Обходим все ячейки карты Карно в порядке возрастания количества единиц (для 2 переменных это $00 \to 10 \to 01 \to 11$). Для очередной ячейки:
   - Если равна 0, идем к следующей;
   - Если равна 1, записываем в полином член, инвертируем все ячейки, где единицы совпадают с единицами ячейки (нарпирмер, если ячейка 010 равна 1, то флипаем $011, 110, 111$ и саму 010).

A \ BC	00	01	11	10
0	①	0	0	1
1	1	0	1	0

A \ BC	00	01	11	10
0	0	1	1	0
1	⓪	1	0	1

A \ BC	00	01	11	10
0	0	1	1	⓪
1	0	1	0	1

A \ BC	00	01	11	10
0	0	①	1	0
1	0	1	0	1

A \ BC	00	01	11	10
0	0	0	0	0
1	0	0	1	①

A \ BC	00	01	11	10
0	0	0	0	0
1	0	⓪	0	0

A \ BC	00	01	11	10
0	0	0	⓪	0
1	0	0	0	0

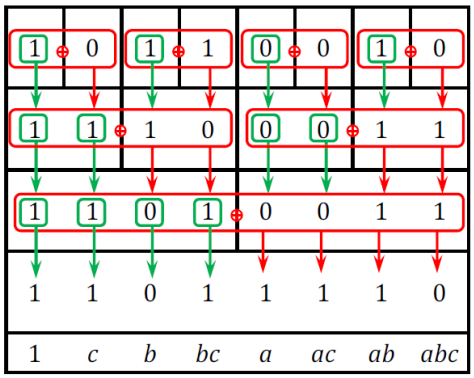
A \ BC	00	01	11	10
0	0	0	0	0
1	0	0	⓪	0

$P = 1 \oplus C \oplus AB$

4. Метод быстрого преобразования Фурье.

   Делаем таблицу $(n + 1) \times 2^n$. В каждой строке делаем сначала блоки по 1 ячейке, потом по 2, по 4, и так до конца. Левые блоки (зеленого цвета) вставляются вниз как есть, правые блоки XOR'ятся с левыми и вставляются вниз. В полином выписываются члены из нижней строки.

   Короче сложно на словах кратко объяснить, так что методом пристального взгляда на картинку:



$f(a,b,c) = 1 \oplus a \oplus c \oplus ab \oplus ac \oplus bc$

⊕ — побитная операция «Исключающее ИЛИ»

Gray code

Код Грея - двоичная кодировка, где соседние значения последовательности отличаются в одном бите.

Пример: $000, 001, 011, 010, 110, 111, 101, 100$

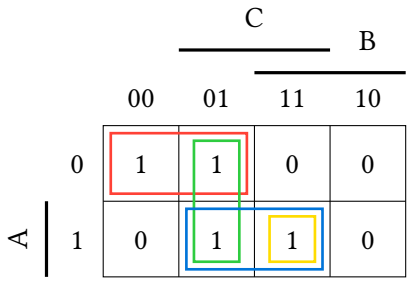
Literal, clause, and cube

- Литерал - переменная или его отрицание (x, $\neg y$)
- Куб - конъюнкция литералов ($x \wedge y \wedge z$)
- Клоз - дизъююнкция литералов ($x \vee y \vee z$)

Implicant, prime implicant, essential prime implicant

- Импликанта - куб в ДНФ или клоз в КНФ
- Простая импликантиа - импликанта, которая не может быть покрыта более общей импликантой.
- Ядровая (essential) простая импликанта - простая импликанта, которая содержит минтерм, которая не покрывается другой импликантой.

На примере карты Карно:



- Красный, зеленый, синий желтый - импликанты
- Красный, зеленый, синий - простые имликанты
- Красный, синий - ядровые простые импликанты

Karnaugh map (K-map) minimization

Карта Карно - представление таблицы истинности в виде двумерной таблицы, упорядоченной по кодам Грея (из-за чего соседние ячейки отличаются только в одной переменной).

Карты Карно используются для ручной визуальной минимизации ДНФ/КНФ.

Пусть мы хотим сделать минимальную ДНФ.

Собственно, мы переводим таблицу истинности в карту Карно, группируем клетки с единицами в прямоугольники длиной в степени двойки (если нужно, то с wrap-around и наложением групп). Потом выписываем термы по правилу:

- Если в каждой ячейке группы в соответсвующем бите стоит единица - выписываем литерал.
- Если в каждой ячейке группы стоит нуль - выписываем отрицание литерала.
- Если в бит различается в клетках - его не выписываем.

Метод карт Карно позволяет быстро минимизировать функцию с маленьким количеством переменных. Чем больше переменных, тем более сложный wraparound. А еще количество клеток растет экспоненциально.

Ну крч бла бла бла, сами нормально объясните; в дз миллион этих карт было.

Don’t-care conditions in K-maps

Don’t-Care условия - ситуации, где выходное значение функции не важно.

Например, если мы точно знаем, что какой-то вход никогда не будет подаваться в функцию, то это Don’t-Care.

В карте Карно на месте Don’t-Care ставим крестик, и используем их в группировке термов по стандартным правилам.

Quine-McCluskey algorithm

Систематизированный табличный алгоритм минимизации.

Алгоритм разбит на две части:

Часть 1: Генерация всех простых импликантов:

1. Выписываем все минтермы в бинарном формате
2. Группируем их по количеству единиц
3. Комбинируем пары минтермов, которые отличаются в 1 бите и заменяем их на минус
4. Повторяем с шаги 2-4, пока есть пары

Часть 2: Выбор минимального множества простых импликантов, покрывающего все минтермы.

1. Делаем таблицу простых импликантов
2. Находим среди них ядровые
3. Добиваем оставшиеся непокрытые минтермы (например, методом Петрика)

Польза алгоритма:

- Его можно запрогать
- Гарантированно выдает минимальную формулу
- Удобен для произвольного количества переменных

Petrick's method for exact cover

Метод Петрика позволяет найти все комбинации простых импликантов, которые покроют оставшиеся минтермы.

1. Берем таблицу простых импликантов из Квайна-Маккласки
2. Перебираем все комбинации импликантов, чтобы покрыть каждый минтерм
3. Записываем импликанты в виде КНФ
4. Переводим в ДНФ
5. Среди всех ДНФ выбираем наименьший

То есть, если вкратце, мы тупо перебираем все возможные покрытия и среди них находим наименьшее.

Superposition (composition) and functional closure

Суперпозиция (композиция) булевых функций - функция, выраженная с помощью других функций из некоторого множества. Например, функцию $\vee$ можно собрать из $\wedge$ и $\neg$.

Функциональное замыкание – множество всех возможных суперпозиций для некоторого множества функций *(базиса, получается?)*.

Functional completeness

Множество Булевых функций F называется *функционально полным*, если, если его замыкание $[F]$ содержит все возможные Булевы функции.

То есть, любая функция может быть выражена, используя только функции из F.

Примеры F:
- $\{\neg, \wedge, \vee\}$
- $\{\text{NAND}\}$
- $\{\wedge, \oplus, \top\}$ *(полином Жегалкина)*

Post's criterion and Post's classes (T0, T1, S, M, L)

Классы Поста:
- T_0 - сохранаяющие 0: $f(0, ..., 0) = 0$
- T_1 - сохраняющие 1: $f(1, ..., 1) = 1$
- S - самодвойственные: $f(\overline{x_1}, ...\overline{x_n}) = \overline{f(x_1, ..., x-n)}$
- M - монотонные: $x < y \Rightarrow f(x) \leq f(y)$
- L - линейные: АНФ степени ≤ 1

Критетрий Поста: F функционально полное $\Leftrightarrow$ для каждого класса Поста есть функция из F, которая не принадлежит этому классу.

Почему именно эти классы? Потому что суперпозиция функций какого-то из данных классов также принадлежит данному классу.

Из-за этого, если все функции из F принадлежат какому-то из данных классов, то $[F]$ содержит не все Булевы функции.

Я думаю это необязательно для теормина, но мне было интересно, почему ваще именно эти классы. В слайде 164 описана интуиция данного критерия.

Sheffer stroke and Peirce arrow

- Штрих Шеффера: NAND, $\uparrow$, $\overline{a \cdot b}$
- Стрелка Пирса: NOR, $\downarrow$, $\overline{a + b}$

Эти операторы функционально полные сами по себе.

SAT

Boolean satisfiability (SAT)

Задача Булевой выполнимости (SAT): существует ли такой набор значений переменных, чтобы формула стала истинной?

Satisfiable, unsatisfiable, valid (tautology)

- Формула φ выполнима, если существует вход, который выдает 1
- Формула φ невыполнима, если все входы выдают 0
- Формула φ валидна (тавтология), если все входы выдают 1

SAT-VALID duality

Формула φ – тавтология $\Leftrightarrow$ $\neg\varphi$ невыполнимо.

2-SAT

Частный случай SAT для КНФ, где каждый клоз имеет ровно 2 литерала.

Суть в том, что мы переводим КНФ в конъюнкцию импликаций, строим из них граф.
- Если в компоненте связности есть литерал и его отрицание – UNSAT.
- Если в каждой компоненте такого нет – SAT.

Complexity classes P, NP, NP-hard, NP-complete

Класс сложности P - класс задач, решаемых за полиномиальное время. Например, сортировки или кратчайшие пути.

Класс сложности NP - класс задач, в которых ответ "да" может быть *проверен* за полином.

Задача X считается NP-сложной, если каждую задачу из NP можно свести к X за полином.

Задача X считается NP-полной, если она принадлежит NP и является NP-сложной.

Cook-Levin theorem

SAT – NP-полная задача.

Это значит, что любую задачу из NP можно свести к SAT за полином.

Tseitin transformation

Алгоритм, позволяющий преобразовать формулу в КНФ за линейное время путем введения дополнительных переменных.

Суть такая:
1. Строим формулу в виде дерева, где вершины - операторы, листья - переменные
2. Для каждого поддерева снизу вверх (кроме листьев) вводим переменную (напр. $t_1 \leftrightarrow (a \vee b)$, $t_2 \leftrightarrow (c \wedge d)$, $t_3 \leftrightarrow (t_1 \to t_2)$)
3. Конъюнктируем все t_i
4. Преобразовываем в КНФ

Resolution

Правило резолюции: если имеем клозы $(A \vee x)$ и $(B \vee \neg x)$, то можно вывести клозу $(A \vee B)$.

Идея в том, что если при повторении этого правила мы получаем пустую клозу, то вся формула UNSAT.

Данное правило используется в программных системах доказательств теорем.

Там еще есть понятия "ширина резолюции" и "размер резолюции", но я чет не особо понял нафига они нужны. Ну, они разве что связаны с асимптотикой резолюции всей формулы.

Unit propagation

Unit propagation - правило упрощения КНФ: если КНФ содержит клозу из 1 литерала, то мы мы во всех клозах убираем этот литерал (в итоге эта 1-клоза удаляется). Отрицания этого литерала тоже удаляем.

В итоге, если мы сократили КНФ до $\top$, то SAT.

Pure literal elimination

Литерал l называется *чистым*, если он встречается в формуле, но $\neg l$ нет.

Правило чистого литерала: если литерал l – чистый, то мы устанавливаем $l = 1$ и удаляем все клозы, содержащие l.

Таким образом, мы сокращаем формулу для SAT солвера.

DPLL algorithm

DPLL - алгоритм для SAT.

Суть такая:
1. Сокращаем формулу: выполняем Unit Propagation и Pure Literal Elimination
2. Если сократили формулу до $\top$, то SAT; если есть пустуая клоза, то UNSAT.
3. Подставляем значение какой-нибудь переменной и повторяем алгоритм.

Визуально это выглядит как дерево, где вершины - подстановки переменных, а листья - $\top$ или $\bot$. Если нашли лист $\top$, то SAT. Если все листья $\bot$, то UNSAT.

Conflict-Driven Clause Learning (CDCL)

CDCL - улучшенная версия DPLL.

Суть в том, что если мы пришли в $\bot$ ("случился конфликт"), то анализируем причину "конфликта" и добавляем ограничительную клозу, которая "запретит" некоторые пути ("чтобы не наступать на те же грабли").

Formal Logic

Propositional logic

Логика высказываний - простейший вид формальной логики, которая работает с высказываниями (propositions), которые могут быть либо истинными, либо ложными.

Также известна как *логика утверждений* или *логика нулевого порядка.*

Interpretation and valuation

*Наверное Костя имел в виду "Interpretation and **e**valuation".*

Интерпретация (или valuation) - функция, которая маппит переменную из высказывания в значение:

$$\nu : V \to \mathbb{B}$$

V - множество переменных. Например для высказывания $A \to (B \vee C)$, $V = \{A, B, C\}$.

$\mathbb{B} = \{0, 1\}$.

Evaluation - рекурсивная интерпретация целой формулы. Valuation же, в свою очередь, интерпретация лишь конкретных атомов.

Logical equivalence

Две формулы φ и ψ называются логически эквивалентными, если все значения истинности совпадают во всех возможных интерпретациях:

$$\varphi \equiv \psi \iff \forall \nu. [\![\varphi]\!]_\nu = [\![\psi]\!]_\nu \iff \models \varphi \iff \psi$$

А еще есть теоремка: "$\varphi \equiv \psi$ т. и т.т, когда $\varphi \Leftrightarrow \psi$ - тавтология"

Semantic entailment

Множество формул Γ *семантически подразумевает* формулу φ, если каждая интерпретация, выполняющая все формулы из Γ, также выполняет формулу φ:

$$\Gamma \vDash \varphi \iff \forall \nu.(\forall \psi \in \Gamma.[\![\psi]\!]_\nu = 1) \to [\![\varphi]\!]_\nu = 1$$

Разбор этой формулы:

1. Перебираем все интерпретации (по таблице истинности, например).
2. Для каждого шага 1, перебираем каждую формулу из Γ. Назовем ее ψ.
3. Для каждого шага 2, проверим, что интерпретация ψ равна 1.
4. Если в переборе 1-2 все истинно, то проверяем интерпретацию φ. Если и она равна 1, то ништяк, $\Gamma \vDash \varphi$.
5. В обратную сторону тоже работает: если мы каким-то образом знаем, что $\Gamma \vDash \varphi$, то шаги 1-4 точно сработают.

Semantic deduction theorem

$$\Gamma \cup \{\varphi\} \vDash \psi \iff \Gamma \vDash \varphi \to \psi$$

Formal proof system (axioms and rules)

Формальная система доказательств состоит из:

1. Набора *аксиом* - формул, принимаемых как всегда истинные
2. Набора *правил вывода* - правил, с помощью которых мы можем вывести новую формулу из уже существующих. Правило состоит из:
   - Предпосылок (премизов)
   - Вывода

Modus ponens and Modus tollens

Modus ponens: $A \to B,\ A\ \therefore B$

Modus tollens: $A \to B,\ \neg B\ \therefore \neg A$

Natural deduction

Естественная дедукция - метод доказательства, не использующий аксиом. Вместо этого, есть правила introduction и elimination - правила “соединения” и “исключения” двух формул.

Например, если мы знаем, что $A \land B$ истинно, можно сделать вывод A и вывод B. (так называемый $\land$ elimination)

Или, например, если A, то можно сделать вывод $A \lor B$. (так называемый $\lor$ introduction)

Fitch notation

Нотация Фитча - способ записи естественной дедукции.

Пример:

1	$C \to B$	Premise
2	$D \lor C$	Premise
3	$\neg B$	Premise
4	$D \to B$	Premise
5	$\neg C$	MT, 1, 3
6	$\neg D$	MT, 4, 3
7	$\quad D$	Assume
8	$\quad \bot$	¬E, 6, 7
9	$\quad C$	Assume
10	$\quad \bot$	¬E, 5, 9
$\therefore$	$\bot$	∨E, 2, 7–8, 9–10

- Нумеруем каждую строку
- Над чертой пишем премизы или предположения
- Под чертой пишем следствия
- Справа от выражений пишем правило вывода

Soundness and Completeness (propositional)

Система доказательств *корректна* (sound), если каждая выводимая формула семантически валидна.

Система доказательств *полна* (complete), если каждая семантически валидная формула формула выводима.

Естественная дедукция является одновременно и sound, и complete.

Categorical propositions (A, E, I, O)

- A – Universal Affirmative:

  $$\forall x.(S(x) \to P(x))$$

- E – Universal Negative

  $$\forall x.(S(x) \to \neg P(x))$$

- I – Particular Affirmative

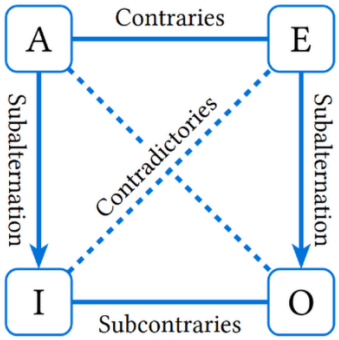
  $$\exists x.(S(x) \to P(x))$$

- O – Particular Negative

  $$\exists x.(S(x) \to \neg P(x))$$

Square of Opposition

Квадрат оппозиции - диаграмма, показывающая логическое отношение между категориальными препозициями:



Ну типа эта диаграмма показывает всякие выводы категориальных пропозиций:

- A-E взаимо противоречивы
- I-O могут быть истинны одновременно
- A-O, E-I не могут быть истинны одновременно
- Если A истинно, то I тоже
- Если E истинно, то O тоже

Existential import

Категориальная пропозиция “S is P” имеет existential import, если оно подразумевает существование хотя бы одного объекта из множества S.

В традиционной логике подразумевается, что все пропозиции имеют existential import.

В современной логике рассматривается существование объекта. То есть, если объекта не существует, то вакуумная истина.

Рассмотрим высказывание: “Все единороги волшебны”.

В традиционной логике это высказывание ложно, поскольку оно подразумевает существование единорогов.

В современной логике это высказывание вакуумно истинно. Типа единорогов не существует, значит из этого можно сделать вообще любой вывод о единорогах (левая часть импликации ложна, значит правая часть неважна, так как вся импликация уже истинна).

Syllogism (Mood and Figure)

Категориальный силлогизм - форма мышления с тремя категориальными высказываниями:

- Мажор премиз - предикат (“все люди смертны”)
- Минор премиз - субъект (“Сократ человек”)
- Вывод (“Сократ смертный”)

Еще есть промежуточный терм, который логически связывает мажор и минор премизы (“человек”).

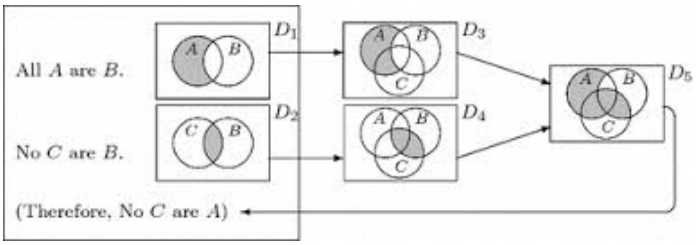
Mood (модус) описывает типы премизов (A-E-I-O).

Figure (фигура) как бы описывает логические переходы: положение промежуточного терма между премизами.

Venn diagrams for syllogistic validity

Ну типа мы можем визуально показать истинность каких-то дедукций (силлогизмов) с помощью диаграмм Венна.

Вот картинка, может по ней будет понятно (я не понял):



First-order logic

Логика первого порядка (логика предикатов) – расширение пропозиционной логики. Добавляются:

- Предикаты (маппинг из объекта в 0/1)
- Кванторы ($\forall$, $\exists$)
- Термы (выражения, описывающие объекты): переменные, константы, функции

Идея в том, что в логике нулевого порядка мы рассматриваем высказывания. В логике первого порядка мы также учитываем модель, внутри которой мы делаем какие-то утверждения.

Ну то есть в FOL какое-то выражение может быть истинно в одной модели, при этом быть ложным в другой. В нулевой логике мы рассматриваем высказывания без учета модели. Ну крч понятно да.